Mikbug ® Operating System

One of the outstanding features of the SWTPC 6800 Microprocessor/System is its use of Motorola's Mikbug ® firmware package stored in the MCM6830L7 ROM. It is thru the software stored in this ROM that the user may use the control terminal to initiate various functions such as load programs, execute programs, dump programs and print or change the contents of the CPU registers. Motorola's Engineering Note 100 describes in detail the operation of the Mikbug ® ROM, however, there are some additional things which should be noted before reading this note.

Also contained within the Mikbug ® ROM is a program called Minibug ® and a section called test pattern. The Minibug ® program is an old and less elaborate version of what is now called Mikbug ® and it along with the test pattern have been disabled from ROM access. The ROM was designed so that either Mikbug ® or Minibug ® could be used and since the MIKBUG ® was by far superior, it was chosen for our system. Although Engineering Note 100 has instructions for the operation of Minibug ® , they should not be read in order to avoid confusion.

Another interesting note is that although Mikbug ® is used with an MC6810 128_{10} word Random Access Memory (RAM) for temporary storage, there are at least 54_{10} RAM locations that are unassigned located from address A$\emptyset$4A$_{16}$ to A$\emptyset$7F inclusive which may be used for user program use. There are also 46_{10} locations from address A$\emptyset$14 to A$\emptyset$41 inclusive which have been reserved for the push-down stack. This means that a 46_{10} deep push down stack may be maintained in the MC6810 RAM or that if a short stack is used, the lower portion of these 46_{10} locations may be used for user program space. It is a good idea, however, not to use any locations between A$\emptyset$37 and A$\emptyset$49 inclusive.

When you start reading about the "Display Contents of MPU Registers Function" in Engineering Note 100 it should be pointed out that the system always assumes the push-down stack is located between address locations A$\emptyset$43 and A$\emptyset$49, with the program counter stored in locations A$\emptyset$48 and A$\emptyset$49. Whenever you enter a "G" for the Go to User's Program Function" the first thing the processor does is execute a "return from interrupt" instruction (RTI) which loads the data stored in address locations A$\emptyset$43 to A$\emptyset$49 into the processor's condition code register, accumulators, index register and program counter. If you forget to use the "Memory Examine and Change Function" to load the program counter locations (A$\emptyset$48 and A$\emptyset$49) with the starting address of your loaded program, or for that matter any of the processor registers which must be initialized to some value, your program of course will not run. At the completion of the "return from interrupt" instruction (RTI) the processor jumps to the starting address of your program. The stack pointer is set to A$\emptyset$49 which means any interrupt, branch to subroutine, jump to subroutine or push instruction in your program will change the data stored at location A$\emptyset$49 and depending upon the instruction, A$\emptyset$48, A$\emptyset$47, etc.. This means that if you abort the program with the RESET button, you will probably have to go back and reload pertinent data in locations A$\emptyset$43 thru A$\emptyset$49 before restarting the program using the "Go to User's Program Function" again. If all of this seems confusing, reread this paragraph after reading Engineering Note 100 contained in this section of the notebook.

Another important note is that there are two subroutines within the Mikbug ® package which greatly aid the programmer in his control terminal input/output routines. They are called INEEE, address E1AC, and OUTEEE, address

E1D1 whose descriptions follow:

INEEE (E1AC) - The INEEE subroutine should be used to receive incoming characters from the control terminal's keyboard via the control interface (serial), I/O port 1. The subroutine loops within itself until a character is received, at which time it is deposited into the A accumulator in ASCII form. Bit 7, the parity bit of the received data is automatically zeroed out and is not checked in any way for accuracy before it is loaded into accumulator A. The B accumulator and index register are used in the subroutine, however the original data in these locations is stored and then restored at the completion of the subroutine.

OUTEEE (E1D1) - The OUTEEE subroutine should be used to transmit characters out of the control interface (serial), I/O port 1, to the control terminal. The ASCII coded character to be transmitted must be loaded into the A accumulator before the subroutine is called. The entire eight bits are transmitted out as they are loaded into the accumulator. No partiy bit determinations are made by the subroutine. The B accumulator and index register are used in the subroutine, however, the original data in these locations is stored and then restored at the completion of the subroutine.

The simplest way to get to these subroutines is to do a jump to subroutine extended (JSR_{asm}) or (BD_{16}) followed by the address of the subroutine.

Entry back into the Mikbug® control program can be done automatically at the end of your program by inserting a jump to address $E\emptyset E3_{16}$ (JMP_{asm}) or ($7E_{16}$) followed by the address $E\emptyset E3_{16}$.

One final note is that the Motorola Engineering Note 100 does not cover the software interrupt function, which is a tool for debugging problems. It is used as follows:

SOFTWARE INTERRUPT (BREAKPOINT) FUNCTION - This software interrupt function provides you with a method of entering breakpoints into your program. Assume that you are debugging your program and wish to verify that your program has reached a particular program instruction. You can, by using the SWI (software interrupt) instruction, enter a breakpoint at this program instruction's address. To enter the breakpoint you load the instruction at the selected address with a SWI instruction. Now, when the SWTPC 6800 System executes this SWI instruction in the user's program, it returns program control to the MIKBUG® software interrupt routine, prints the contents of the MPU registers, and proceeds to the MIKBUG® control program. This software interrupt mode (SWI) which displays the contents of the MPU registers does not assume the stack is located between address AØ43 and AØ49 like the "Display contents of MPU Registers Function" of the MIKBUG® control program. The register data printout will be accurate no matter where the stack is positioned in memory. The following paragraphs discuss entering the breakpoint into and removing a breakpoint from a user's program.

ENTERING A SWI BREAKPOINT - Use the following procedures to enter a software interrupt breakpoint into the user's program. It is assumed prior to these procedures that the user's program has been loaded into memory, the SWTPC 6800 System is performing its MIKBUG® control program, and the last character printed by the data terminal is an asterisk.

a. Enter a M after the asterisk to open a memory location. The terminal

will insert a space after the M.

b. Enter in 4-character hexadecimal the memory address you have selected to enter a breakpoint. The terminal will print on the next line this memory address and its contents. Record the memory contents.

c. Enter a space code and the hexadecimal characters 3F (SWI instruction). The SWI instruction is now stored in memory and the terminal prints the next address and its contents on the next line.

d. Enter a space code followed by carriage return control character. The SWTPC 6800 System returns to the MIKBUG® control program and the terminal prints an asterisk on the next line.

e. Run tne user's program in accordance with the Go To User's Program Function (described in Engineering Note 100). When the SWTPC 6800 System executes the SWI instruction, it returns program control to the MIKBUG® program, prints the contents of the MPU registers, and advances to the MIKBUG® control program.

REMOVING A SWI BREAKPOINT AND RESTORING THE PROGRAM - Use the following procedures to remove a software interrupt breakpoint from the user's program. It is assumed at the start of these procedures that SWI instruction has been loaded into a known memory location, the SWTPC 6800 System is performing its MIKBUG® control program, and the last character printed by the terminal is an asterisk.

a. Enter a M after the asterisk to open a memory location. The terminal will insert a space code after the M.

b. Enter in 4-character hexadecimal code the address whose SWI instruction is to be removed. The terminal will print on the next line this memory and address and 3F (SWI instruction).

c. Enter a space code and restore the original instruction removed from this address previously. The new contents are stored in memory and the terminal prints the following memory address and its contents on the next line.

d. Enter a space code followed by a carriage return control character. The SWTPC 6800 System returns to the MIKBUG® control program and the terminal prints an asterisk on the next line.

Dual Address Memory Test CDAT
By John Christensen

The CDAT memory diagnostic can be used to help locate memory problems in a SWTPC 6800 computer system that MEMCON and ROBIT may miss. The program itself resides entirely within the 128 byte MIKBUG® RAM. The program must be loaded in two parts to avoid interfering with the systems push down stack. The contiguous section of memory to be tested is set by loading the most significant byte of the lower memory address into A00Ø2, the least significant byte into A00Ø3, the most significant byte of the upper memory address in A00Ø4 and its least significant byte in A00Ø5. The low address must be less than or equal to the upper address.

The test starts from the low address and writes a ØØ into all memory up to the high address. An FF is then written into the first address and all other locations are checked to be sure they contain ØØ. If all are OK the FF is replaced with a ØØ and an FF is written in the next memory loaation. This pattern continues until all memory is checked or an error is found. If the computer returns to MIKBUG®, then no errors were found.

If the program displays a register dump then a problem was discovered on the board. The register dump should look similar to the following:

```
                ADDRESS→   ←ERROR MSG.
F3  ØØ  FF  Ø4ØØ  AØ79  AØ42
```

The important parts of the dump are the ADDRESS and the ERROR MSG. areas, as denoted above. The error messages are interpreted as follows:

AØ77	Error on initial test pattern (can't write Ø's into mem.)
AØ78	Error on second test pattern (can't write FF's into mem.)
AØ79	Dual address error low
AØ7A	Dual address error high

If a dual address error is found then writing into one memory location affects another. For example, if ADDRESS = Ø4ØØ and AØ16 contains Ø41Ø then writing into Ø4ØØ will change the contents of Ø41Ø or vice-versa. (AØ16 is a temporary index register storage location within the program that you can compare with ADDRESS in the register dump to see which two memory locations caused the error). The IC assignments table included with the memory board instructions can then be used to help locate the problem by comparing the bit pattern of the locations in error.

The CDAT program takes some time to run, so run the diagnostic over only one complete board at a time.

MEM. SIZE	APPROX. RUN TIME
1K	29 sec.
2K	1 min. 53 sec.
3K	4 min. 13 sec.
4K	7 min. 29 sec.
8K	more than 30 min.

MIKBUG® is a registered trademark of Motorola, Inc.

```
                        NAM     CDAT1
                    *MEM DIAGNOSTIC ( JOHN CHRISTENSEN '8 )
                        OPT     O
     E0E3       CONTRL EQU     $E0E3
A002                    ORG     $A002
A002 0002       LOTEMP RMB     2           STARTING ADDRESS
A004 0002       HITEMP RMB     2           ENDING ADDRESS


A014                    ORG     $A014
A014 00         INIPAT FCB     0           INITIAL TEST PATTERN
A015 FF         TESPAT FCB     $FF         TEST PATTERN
A016 0002       IXRTMP RMB     2           IXR TEMPORARY STORAGE
A018 FE A002    START  LDX     LOTEMP
A01B B6 A014           LDA A   INIPAT
A01E A7 00      LOOP1  STA A   0, X
A020 A1 00             CMP A   0 X
A022 26 53             BNE     ERPNT1
A024 BC A004           CPX     HITEMP
A027 27 03             BEQ     LOAPAT
A029 08                INX
A02A 20 F2             BRA     LOOP1
A02C FE A002 LOAPAT    LDX     LOTEMP
A02F F6 A015           LDA B   TESPAT
A032 E7 00      LOOP4  STA B   0, X
A034 20 14             BRA     CHECK


A048                    ORG     $A048
A048 A018               FDB     $A018
A04A E1 00      CHECK  CMP B   0, X
A04C 26 2A             BNE     ERPNT2
A04E FF A016 CHKLOW    STX     IXRTMP
A051 BC A002 LOOP2     CPX     LOTEMP
A054 27 07             BEQ     CHCKHI
A056 09                DEX
A057 A1 00             CMP A   0, X
A059 26 1E             BNE     ERPNT3
A05B 20 F4             BRA     LOOP2
A05D FE A016 CHCKHI    LDX     IXRTMP
A060 BC A004           CPX     HITEMP
A063 27 16             BEQ     END
A065 08         LOOP3  INX
A066 A1 00             CMP A   0, X
A068 26 10             BNE     ERPNT4
A06A BC A004           CPX     HITEMP
A06D 26 F6             BNE     LOOP3
A06F FE A016 RESTRE    LDX     IXRTMP
A072 A7 00             STA A   0, X
A074 08                INX
A075 20 BB             BRA     LOOP4
A077 3F         ERPNT1 SWI                 ERROR ON INITIAL PATTERN
A078 3F         ERPNT2 SWI                 ERROR ON TEST PATTERN
A079 3F         ERPNT3 SWI                 DUAL ADDRESS ERROR LOW
A07A 3F         ERPNT4 SWI                 DUAL ADDRESS ERROR HI
A07B 7E E0E3    END    JMP     CONTRL
                       END
```